Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm

Jennifer de Kleine
dekleine@cecm.sfu.ca *

Michael Monagan
mmonagan@cecm.sfu.ca *

Allan Wittkopf
awittkop@maplesoft.com *

Centre for Experimental and Constructive Mathematics
Simon Fraser University,
Burnaby, British Columbia, V5A 1S6 Canada

Abstract

Let G be the greatest common divisor (GCD) of two polynomials $A, B \in \mathbb{Z}[x, y, z]$ where x is the main variable. Suppose $G = (4y^2 + 2z)x^2 + (5y^2 + 3z)$. Because G is not monic in x the sparse modular GCD algorithm of Richard Zippel cannot be applied directly as one is unable to scale the univariate images of G in x consistently. We call this the *normalization problem*. It can be solved in a number of ways. The approach taken by Paul Wang is to factor the leading coefficient $L(y, z)$ of one of the input polynomials over $\mathbb{Z}$ and determine which factors of L belong with G. If A or B is sparse then one can usually select a main variable and leading coefficient that is easy to factor. If A and B are not sparse, the factorization could be expensive.

We present two new sparse modular GCD algorithms which do not require any factorization. The first algorithm is a modification of Zippel's algorithm where the scaling factors are treated as unknowns to be solved for. This leads to a structured coupled linear system for which an efficient solution is still possible. The second algorithm reconstructs the monic GCD $x^2 + (10y^2 + 6z)/(2y^2 + z)$ from monic univariate images using a sparse, variable at a time, rational function interpolation algorithm. We present an initial run-time comparison of a Maple implementation of the two methods on three classes of GCD problems.

1 Introduction

Let A, B be polynomials in $\mathbb{Z}[x_1, ..., x_n]$. Let G be their greatest common divisor (GCD) and let $\bar{A} = A/G$, $\bar{B} = B/G$ be their cofactors. Our problem is to compute G, $\bar{A}$ and $\bar{B}$. It was Brown in [1] who first presented an efficient solution. Brown's algorithm reconstructs $G, \bar{A}$ and $\bar{B}$ from a sequence of univariate images of G. It does this by reducing the inputs modulo a sequence of primes of near constant bit length and then successively, evaluating out all variables except x_1, the main variable. If d bounds the degree of A and B in all variables, then Brown's algorithm requires $\mathcal{O}(d^{n-1})$ univariate GCD computations for each prime.

Consider $G = x_1^d + x_2^d + ... + x_n^d - 1$. For this problem Brown's algorithm will perform at least d^{n-1} univariate GCD computations even though G has only $n + 1$ non-zero terms. In [13] (see also [15] for a more accessible reference) Zippel presented a Las Vegas algorithm for computing G when G is monic which improves on the running time of Brown's algorithm when G is also sparse. If t is the maximum number of terms of a coefficient of G in x_1 (in the example $t = n$), Zippel's algorithm

*This work was supported by the MITACS NCE of Canada and NSERC of Canada

requires only $\mathcal{O}((n-1)td)$ univariate GCD computations on average. Zippel's algorithm is also an *output sensitive* algorithm. Unlike Brown's algorithm, the number of univariate GCD computations depends on the size of G, only, and not also on $\bar{A}$ nor $\bar{B}$. The improvement obtained in practice is often dramatic for problems with many variables.

Zippel's algorithm has been implemented in Macsyma, Magma, and Mathematica. A parallel implementation of the algorithm is described by Rayes, Wang and Weber in [11]. Previous work done to improve the asymptotic efficiency of the algorithm includes that of Zippel in [14], and Kaltofen and Li in [5, 6].

In this paper we present two new approaches for extending Zippel's algorithm to the case where G is not monic in the main variable. In section 2 we give a description of Zippel's algorithm and previous approaches to extend it to the non-monic case. In section 3 we describe our first solution and in section 4 our second solution. In section 5 we describe our Maple implementation of the two algorithms and present a run time comparison of the two algorithms on three classes of GCD problems. We end with some remarks about further improvements. Although our algorithms do not require any polynomial factorizations, both require the content of G in the main variable x_1 to be computed and removed from the inputs. We describe how to do this in Appendix A.

2 Zippel's Algorithm

There are two subroutines in the algorithm, subroutines M and P. Subroutine M, the main subroutine, computes $G = \text{GCD}(A, B)$ where $A, B \in \mathbb{Z}[x_1, ..., x_n]$. It does this by computing $\text{GCD}(A, B)$ modulo a sequence of primes $p_1, p_2, ...$ and it reconstructs G from these images by applying the Chinese Remainder Theorem. The first image G_1 is computed by calling subroutine P with inputs $A \bmod p_1$ and $B \bmod p_1$.

Subroutine P, which is recursive, computes $G = \text{GCD}(A, B)$ where A, B in $\mathbb{Z}_p[x_1, ..., x_n]$ for some prime p as follows. If $n = 1$ it uses the Euclidean algorithm. If $n > 1$ it computes $\text{GCD}(A, B)$ at a sequence of random points $\alpha_1, \alpha_2, ... \in \mathbb{Z}_p$ for x_n and reconstructs $G \in \mathbb{Z}_p[x_1, ..., x_n]$ from the images using *dense* interpolation, e.g., using Newton interpolation. The first image G_1 is computed by calling subroutine P recursively with inputs $A \bmod \langle x_n - \alpha_1 \rangle$ and $B \bmod \langle x_n - \alpha_1 \rangle$.

In both subroutines, after the first image G_1 is computed, subsequent images are computed using *sparse* interpolations. This involves solving a set of independent linear systems which are constructed based on the form of G_1. The main assumption is that G_1 is of the correct form, that is, all non-zero terms of the GCD G are present. This is likely true if the primes are sufficiently large and the evaluation points are chosen at random from $\mathbb{Z}_p$. If it is not true then the algorithm needs to detect this. Let's work through an example. Our examples use x for the main variable and y, z for non-main variables.

Example 1 *Consider the computation of the bivariate* GCD $G = x^2 + 3y^3x + 35 \in \mathbb{Z}[x, y]$, *with input polynomials* $A = (y+1)G$ *and* $B = (x+1)G$. *Suppose* $p_1 = 11$. *We call subroutine P and compute our first* GCD *image* $G_1 \in \mathbb{Z}_{11}[x, y]$ *as follows. First we compute* d_y *a degree bound on* $\deg_y(G)$. *We set* $d_y = deg_y(\text{GCD}(A(1, y), B(1, y)) \bmod 11) = 3$. *Next, we compute* $d_y + 1$ *monic univariate* GCD *images at random points for* y *using the Euclidean algorithm. Suppose we use* $y = 2, 4, 5, 6$. *We obtain*

$$\begin{aligned}
g_1 = \text{GCD}(A(x,2),\ B(x,2)) \bmod 11 &= x^2 + 2x + 2 \pmod{11}\\
g_2 = \text{GCD}(A(x,4),\ B(x,4)) \bmod 11 &= x^2 + 5x + 2 \pmod{11}\\
g_3 = \text{GCD}(A(x,5),\ B(x,5)) \bmod 11 &= x^2 + x + 2 \pmod{11}\\
g_4 = \text{GCD}(A(x,6),\ B(x,6)) \bmod 11 &= x^2 + 10x + 2 \pmod{11}.
\end{aligned}$$

Now we interpolate y to construct our first bivariate image. We obtain $G_1 = x^2 + 3y^3x + 2 \pmod{11}$. Our assumed form of the GCD *G in subroutine M is*

$$G_f = x^2 + \alpha y^3 x + \beta.$$

Notice that if we had used $p_1 = 3, 5$ or 7, a coefficient of G would vanish and we would not get the correct assumed form. Let $p_2 = 13$. We compute a second GCD *image in $\mathbb{Z}_{13}[x, y]$ using a sparse interpolation based on the assumed form G_f. We have at most one unknown per coefficient in our main variable x so we need only one evaluation point. We choose $y = 8$, and obtain the univariate* GCD *image $x^2 + 2x + 9$. We evaluate G_f at our chosen point and equate, to get:*

$$x^2 + 5\alpha x + \beta \quad = \quad x^2 + 2x + 9.$$

Solving the two independent systems $\{5\alpha = 2\}$ and $\{\beta = 9\}$ modulo 13 gives $\alpha = 3$ and $\beta = 9$ resulting in the new image $G_2 = x^2 + 3y^3x + 9 \pmod{13}$. We now apply the Chinese Remainder Theorem to the integer coefficients of G_1 and G_2 to reconstruct our GCD *over $\mathbb{Z}$.*

Not all primes and evaluation points can be used in the algorithm. We will identify three cases here indicating three ways the algorithm can go wrong and hence three classes of primes and evaluation points that must be detected.

Definition 1 (bad prime and bad evaluation point.) *A prime p in subroutine M is bad if $\deg_{x_1}(\text{GCD}(A \bmod p, B \bmod p)) < \deg_{x_1}(G)$. Similarly, an evaluation point $(\alpha_1, ..., \alpha_{n-1}) \in \mathbb{Z}_p^{n-1}$ is bad if $\deg_{x_1}(\text{GCD}(A \bmod I, B \bmod I)) < \deg_{x_1}(G)$ where $I = \langle x_2 - \alpha_1, ..., x_n - \alpha_{n-1}\rangle$.*

For example, if $A = (3yx + 1)(3x - y)$ and $B = (3yx + 1)(x - 3y)$ then 3 is a bad prime and $y = 0$ is a bad evaluation point. Bad primes and bad evaluation points must be avoided so that the univariate images can be scaled consistently.

Definition 2 (unlucky prime and unlucky evaluation point.) *A prime p is unlucky if the cofactors are not relatively prime modulo p, i.e., $\deg_{x_1}(\text{GCD}(\bar{A} \bmod p, \bar{B} \bmod p)) > 0$. Similarly an evaluation point $(\alpha_1, ..., \alpha_{n-1}) \in \mathbb{Z}_p^{n-1}$ is unlucky if $\deg_{x_1}(\text{GCD}(\bar{A} \bmod I, \bar{B} \bmod I)) > 0$ where $I = \langle x_2 - \alpha_1, ..., x_n - \alpha_{n-1}\rangle$.*

For example, if $\bar{A} = 7x + 6y$ and $\bar{B} = 12x + y$ then $p = 5$ is an unlucky prime and $y = 0$ is an unlucky evaluation point. Unlucky primes and evaluation points must be avoided if G is to be correctly reconstructed. Unlike bad primes and bad evaluation points, they cannot be ruled out in advance. Instead they identify themselves when we encounter a univariate image in x_1 of higher degree than other univariate images.

Definition 3 (missing terms.) *A prime p is said to introduce missing terms if any integer coefficient of G vanishes modulo p. Similarly, an evaluation $x_n = \alpha$ (in subroutine P) is said to introduce missing terms if any coefficient in $\mathbb{Z}_p[x_n]$ of G vanishes at $x_n = \alpha$.*

In example 1 where GCD $G = x^2 + 3y^3x + 35 \in \mathbb{Z}[x, y]$, the primes $p = 3, 5$ or 7 and the evaluation $y = 0$ cause terms in G to vanish. Zippel's algorithm cannot reconstruct G if it uses such primes and evaluation points for the first image G_1. Such cases are detected in subroutines M and P by trial division. Suppose G' is the reconstructed result. Before terminating in subroutine M and P, the correctness of G' is established by testing if G' divides A and B.

Example 2 *Consider computing the non-monic bivariate* GCD

$$G = (y+50)x^3 + 100y \in \mathbb{Z}[x,y]$$

from input polynomials $A = (y+1)\,G$ and $B = (y+2)\,G$. Here G has leading coefficient $y+50$ in the main variable x. Suppose we compute our first bivariate image modulo $p_1 = 13$ and obtain $G_1 = (y+11)x^3 + 9y \pmod{13}$. We proceed to compute a second image using sparse interpolation working modulo 17. First, assume the GCD *has the form $G_f = (y+\alpha)x^3 + \beta y$ for some $\alpha, \beta \in \mathbb{Z}_{17}$. We have at most one unknown per coefficient in x so we evaluate at one random point, $y = 5$, and compute the univariate* GCD *$x^3 + 6 \pmod{17}$. We evaluate G_f at $y = 5$ and equate to obtain:*

$$(5+\alpha)x^3 + 5\beta \quad = \quad x^3 + 6$$

Solving for α and β in $\mathbb{Z}_{17}$, we obtain the bivariate image $G_2 = (y+13)x^3 + 8y$, which is incorrect. The correct image is $G \bmod 17 = (y+16)x^3 + 15y$. The problem is that at the bottom level we compute a univariate GCD *modulo p which will always be monic. We thus always equate the leading coefficient to be 1, giving us incorrect results. We call this the normalization problem.*

A solution to this problem can be obtained in a number of ways. One approach is to normalize the univariate images by multiplying through by the image of $\gamma = \text{GCD}(\mathsf{lc}_{x_1}(A), \mathsf{lc}_{x_1}(B))$, the GCD of the leading coefficients of the input polynomials. Now $\text{lc}_x(G)$ divides γ hence $\gamma = \Delta \times \mathsf{lc}_x(G)$. If $\Delta = 1$ then this approach works very well. However, it may happen that Δ is a non-trivial polynomial. In that case, we would have to reconstruct $\Delta \times G$ a larger and likely denser polynomial than G. Another disadvantage is that one must compute γ and also compute the content of $\Delta \times G$ both of which require additional multivariate GCD computations.

A better solution is to factor L the (possibly multivariate) leading coefficient of one of the input polynomials, A say, and then determine which factors of L belong with G and (implicitly) which belong with $\bar{A}$. In [9, 10], Wang shows how to do this heuristically (for the *EEZ-GCD* algorithm) for polynomials with integer coefficients. The problem then becomes one of factoring a multivariate polynomial in at least one fewer variable. If the inputs are not dense, the factorization is usually not hard. Kaltofen in [3] shows how to reduce the factorization to a bivariate factorization and how to make Wang's heuristic work for coefficient rings other than $\mathbb{Z}$.

3 Algorithm LINZIP

In Zippel's algorithm, if the leading coefficient of G in x_1 is a monomial in $x_2, ..., x_n$, then there is an easy solution. In fact, if *any* coefficient of the GCD with respect to the main variable x_1 is a monomial then the normalization is straightforward. For example, consider the GCD problem from Example 2. Notice that the $\mathcal{O}(x^0)$ term in our first GCD image $G_1 = (y+11)x^3 + 9y$ has a single term coefficient, $9y$. Since we know the exact form, we can scale our univariate GCD images based on this term. Our assumed form becomes $G_f = (\alpha y + \beta)x^3 + (1)y$ for some $\alpha, \beta \in \mathbb{Z}_{17}$. Now we have two unknowns in our $\mathcal{O}(x^3)$ term so we need two evaluation points, neither of which may be 0. We choose $y = 5, 7$, to get the univariate GCDs $x^3 + 6 \pmod{17}$ and $x^3 + 9 \pmod{17}$, respectively. Now we scale the first univariate GCD by $\frac{5}{6} \pmod{17}$, and the second by $\frac{7}{9} \pmod{17}$ before equating, giving:

$$\begin{aligned}(5\alpha + \beta)x^3 + 5 &= 15x^3 + 5\\ (7\alpha + \beta)x^3 + 7 &= 14x^3 + 7.\end{aligned}$$

Solving for α and β in $\mathbb{Z}_{17}$, gives us the bivariate image, $(8y+9)x^3+y$, which when made monic gives us the correct image $G_2 = (y+16)x^3+15y \pmod{17}$. Thus if (at any level in the recursion) an image has a coefficient in x_1 which is a single term, the normalization problem is easily solved.

The normalization problem essentially reduces to scaling of the univariate GCD images so that the solution of the linear system produces a correct scalar multiple of the GCD. The approach followed now is quite simple in concept, and is to treat the scaling factors of the computed univariate GCDs as unknowns as well. This may result in larger linear systems requiring additional univariate GCD images. We call this the *multiple scaling* case (as opposed to the *single scaling* case).

Scaling of both the univariate GCDs and the coefficients of the assumed form of the multivariate GCD results in a system that is under-determined by exactly 1 unknown (the computation is only determined up to a scaling factor). Rather than fixing an unknown in the form of the GCD to 1, we instead fix the scaling factor of the first GCD to 1. The following example illustrates this approach.

Example 3 *Consider the computation of the bivariate* GCD $(3y^2-90)x^3+12y+100$. *We obtain* $g \equiv x^3y^2+9x^3+4y+3 \pmod{13}$, *and assumed form of the* GCD $g_f = \alpha x^3y^2+\beta x^3+\gamma y+\sigma$. *Instead of computing two univariate* GCD *images for the new prime* $p_2 = 17$, *we compute three, choosing* $y = 1, 2, 3$ *and obtain the* GCD*s* x^3+12, x^3+8, *and* x^3 *respectively. We form the modified system as follows:*

$$\begin{aligned} \alpha x^3+\beta x^3+\gamma+\sigma &= m_1\,(x^3+12) = x^3+12, \\ 4\alpha x^3+\beta x^3+2\gamma+\sigma &= m_2\,(x^3+8), \\ 9\alpha x^3+\beta x^3+3\gamma+\sigma &= m_3\,(x^3), \end{aligned}$$

where m_2, m_3 *are the new scaling factors, and we have set the first scaling factor* m_1 *to* 1. *Solving this system yields* $\alpha = 7, \beta = 11, \gamma = 11, \sigma = 1$, *with scaling factors* $m_2 = 5, m_3 = 6$, *so our new* GCD *image is given by* $g \equiv 7x^3y^2+11x^3+11y+1 \pmod{17}$ *which is consistent with our* GCD.

Now we explain the reason for fixing a univariate GCD multiplier value instead of a coefficient in the assumed form of the GCD. In general it is possible that the chosen prime or evaluation sets the scaling term in the GCD to zero, but there is no way to detect it. In example 3, suppose we set $\alpha = 1$. In this case, the evaluation $y = 0$ is not bad but $\alpha = 0$, so the resulting system will be incorrect. Attempting to set $\beta = 1$ will have the same problem for the prime $p = 5$. In contrast, choosing primes and evaluations so that the leading term of the GCD does not vanish (detected as a bad prime or evaluation) does not exhibit this problem.

One might wonder why the multiple scaling case can be even mildly efficient, as we are constructing a system that ties together all unknowns of the problem through the multipliers. This is in direct contrast to the single scaling case, for which each degree in x_1 has an independent subsystem. The trick is to realize that the resulting system is highly structured, and the structure can be exploited to put the solution expense of the multiple scaling case on the same order as the solution expense of the single scaling case.

Example 4 *Consider the linear system that must be solved to compute the* GCD *for a problem with the assumed form:*

$$g_f = (a_{32}y^2+a_{31}y+a_{30})x^3+(a_{23}y^3+a_{22}y^2+a_{21}y+a_{20})x^2+(a_{12}y^3+a_{11}y+a_{10})x+(a_{01}y^2+a_{00})$$

We require 4 *images to have a sufficient number of equations to solve for all unknowns (we note that for this problem the number of required images is exactly the same as would be required for the single scaling case). The resulting linear system, involving 16 equations for the 15 unknowns (12 image unknowns and 3 unknown scaling factors) has the following structure:*

$$
\left[\begin{array}{cccccccccccccccc}
c & c & c & & & & & & & & & & 1 & & & \\
c & c & c & & & & & & & & & & & 1 & & \\
c & c & c & & & & & & & & & & & & 1 & \\
c & c & c & & & & & & & & & & & & & 1 \\
& & & c & c & c & c & & & & & & c & & & \\
& & & c & c & c & c & & & & & & & c & & \\
& & & c & c & c & c & & & & & & & & c & \\
& & & c & c & c & c & & & & & & & & & c \\
& & & & & & & c & c & c & & & c & & & \\
& & & & & & & c & c & c & & & & c & & \\
& & & & & & & c & c & c & & & & & c & \\
& & & & & & & c & c & c & & & & & & c \\
& & & & & & & & & & c & c & c & & & \\
& & & & & & & & & & c & c & & c & & \\
& & & & & & & & & & c & c & & & c & \\
& & & & & & & & & & c & c & & & & c
\end{array}\right]
\left[\begin{array}{c} a_{32} \\ a_{31} \\ a_{30} \\ a_{23} \\ a_{22} \\ a_{21} \\ a_{20} \\ a_{12} \\ a_{11} \\ a_{10} \\ a_{01} \\ a_{00} \\ 1 \\ m_2 \\ m_3 \\ m_4 \end{array}\right]
=
\left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right]
\qquad (1)
$$

Where the equations are ordered by decreasing degree in x then by image number, the unknowns are in the same order as in the image followed by the scaling factors, the c's denote (possibly) non-zero entries, and the 1's denote value 1 entries, all other entries being strictly zero.

The solution of the above system can be easily computed by solution of a number of smaller subsystems corresponding to the rectangular blocks of non-zero entries augmented with the multiplier columns. Once the subsystems are upper triangular, remaining rows, only involving the multipliers, can be used to compute the multiplier values, which can then be back-substituted into the subsystems to obtain the image coefficients.

Note that for the single scaling case, the structure is similar to the above, except that the scaling columns would be replaced by a single column of constants. For that case, all blocks are clearly independent, making the efficient solution of the problem obvious.

This approach does offer a solution to our problem, but it also introduces another difficulty, as illustrated by the following example:

Example 5 *Consider the computation of the bivariate* GCD $(y+2)x^3+12y^2+24y$. *By a call to algorithm P we obtain our first image as* $g_1 = x^3y+2x^3+12y^2+11y \pmod{13}$, *and assumed form of the* GCD *as* $g_f = \alpha x^3y+\beta x^3+\gamma y^2+\sigma y$.

We require at least three univariate GCD *images for the new prime* $p_2 = 17$. *Choosing* $y = 1, 2, 3$ *we obtain the* GCD*s* x^3+12, x^3+7, *and* x^3+2 *respectively, and form the modified system as follows:*

$$
\begin{aligned}
\alpha x^3+\beta x^3+\gamma+\sigma &= x^3+12, \\
2\alpha x^3+\beta x^3+4\gamma+2\sigma &= m_2(x^3+7), \\
3\alpha x^3+\beta x^3+9\gamma+3\sigma &= m_3(x^3+2),
\end{aligned}
$$

In attempting to solve this system, we find that it is under-determined, so we add a new evaluation point, $y = 4$, *obtaining a* GCD *of* x^3+14, *and the new equation*

$$
4\alpha x^3+\beta x^3+16\gamma+4\sigma = m_4(x^3+14).
$$

In attempting to solve the new system of equations, we find that it is still under-determined. In fact, we could continue to choose new evaluation points for y until we run out of points in $\mathbb{Z}_p$, and the linear system would remain under-determined. This is the case for any chosen prime and set of evaluations, so the algorithm fails to find the GCD *for this problem.*

What is not necessarily obvious from example 5 is the cause of the failure, which is the presence of a content in the GCD with respect to the main variable x, namely $y+2$. The existence of a content in the evaluated variables can be immediately recognized as a source of problems for the algorithm, as the evaluated content can be absorbed into the multipliers on the right hand side of the formed system for the equations with multipliers, and otherwise absorbed into each of the unknowns (for the first equation where the multiplier is set to 1).

This is exactly what occurs in example 5, as the content in y is absorbed into the unknowns, so we are unable to obtain a solution for the coefficients in our candidate form as only the relative ratio between terms can be computed.

It is clear that content can cause a problem for the algorithm, so just as for many other sparse algorithms (*EEZ-GCD* for example), the content *must* be removed before the algorithm is called.

One may ask *is it sufficient to simply remove the* GCD *content before calling the algorithm to take care of this problem?* Unfortunately, the answer is no, as certain choices of primes and evaluation points can cause an *unlucky content* to appear in the GCD.

Definition 4 (Unlucky Content) *Given $a \in \mathbb{Z}[x_1, ..., x_n]$ with $cont_{x_1}(a) = 1$, a prime p is said to introduce an* unlucky content *if $cont_{x_1}(a \bmod p) \neq 1$. Similarly for $a \in \mathbb{Z}_p[x_1, ..., x_n]$ with $cont_{x_1}(a) = 1$, an evaluation $x_i = \alpha_i$ is said to introduce an* unlucky content *if $cont_{x_1}(a \bmod \langle x_i - \alpha_i \rangle) \neq 1$.*

Consider, for example, computation of the GCD $x(y+1) + y + 14$. If we choose $p = 13$ the GCD has a content of $y+1$, while for any other prime, or over $\mathbb{Z}[x, y]$, no content is present. This is a significant consideration in the design of the algorithm. An argument can be made as to the similarity between the probability of obtaining an unlucky content, and the probability of selecting an unlucky prime or evaluation, and the fact that these are both equally unlikely. As a result we will design the algorithm so this problem is not detected in advance, but rather through its effect, so that detection of this problem does not become a bottleneck of the algorithm.

We now present the *LINZIP M* algorithm, which computes the GCD in $\mathbb{Z}[x_1, ..., x_n]$ from a number of images in $\mathbb{Z}_p[x_1, ..., x_n]$, and the *LINZIP P* algorithm, which computes the GCD in $\mathbb{Z}_p[x_1, ..., x_n]$ from a number of images in $\mathbb{Z}_p[x_1, ..., x_{n-1}]$.

Algorithm 1 (LINZIP M)

Input: *$a, b \in \mathbb{Z}[x_1, ..., x_n]$ such that $\text{GCD}(\text{cont}_{x_1}(a), \text{cont}_{x_1}(b)) = 1$ and degree bounds $d_{\mathbf{x}}$ on the* GCD *in $x_1, ..., x_n$*

Output: $g = \text{GCD}(a, b) \in \mathbb{Z}[x_1, ..., x_n]$

1 *Compute the scaling factor $\gamma = \text{GCD}(\text{lc}_{x_1,...,x_n}(a), \text{lc}_{x_1,...,x_n}(b)) \in \mathbb{Z}$*

2 *Choose a random prime p such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$, $b_p = b \bmod p$, then compute from these a modular* GCD *image $g_p \in \mathbb{Z}_p[x_1, ..., x_n]$ with a call to LINZIP P. If the algorithm returns* **Fail**, *repeat, otherwise set $d_{x_1} = \deg_{x_1}(g_p)$ and continue.*

3 *Assume that g_p has no missing terms, and that the prime is not unlucky. We call the assumed form g_f. There are two cases here.*

3.1 *If there exists a coefficient of x_1 in g_f that is a monomial, then we can use single scaling and normalize by setting the integer coefficient of that monomial to 1. Count the largest number of terms in any coefficient of x_1 in g_f, calling this n_x.*

```
    3.2 If there is no such coefficient, then multiple scaling must be used. Compute the minimum
        number of images needed to determine $g_f$ with multiple scaling, calling this $n_x$.
4 Set $g_m = (\gamma_p/\mathrm{lc}_{x_1,...,x_n}(g_p)) \times g_p \bmod p$ and $m = p$.
5 Repeat
    5.1 Choose a new random prime $p$ such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$,
        $b_p = b \bmod p$.
    5.2 Set $S = \emptyset$, $n_i = 0$.
    5.3 Repeat
        5.3.1 Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p \backslash \{0\}$ at random such that for $I = \langle x_2 - \alpha_2, ..., x_n - \alpha_n \rangle$ we have
              $\deg_{x_1}(a_p \bmod I) = \deg_{x_1}(a)$, $\deg_{x_1}(b_p \bmod I) = \deg_{x_1}(b)$, and set $a_1 = a_p \bmod I$,
              $b_1 = b_p \bmod I$
        5.3.2 Compute $g_1 = \mathrm{GCD}(a_1, b_1)$
        5.3.3 If $\deg_{x_1}(g_1) < d_{x_1}$ our original image and form $g_f$ and degree bounds were unlucky,
              so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.
        5.3.4 If $\deg_{x_1}(g_1) > d_{x_1}$ our current image $g_1$ is unlucky, so goto 5.3.1, unless the number
              of failures $> \min(2, n_i)$, in which case assume p is unlucky and goto 5.1.
        5.3.5 For single scaling, check that the scaling term in the image $g_1$ is present. If not, the
              assumed form must be wrong, so goto 2.
        5.3.6 Add the equations obtained from equating coefficients of $g_1$ and the evaluation of
              $g_f \bmod I$ to $S$, and set $n_i = n_i + 1$.
    Until $n_i \geq n_x$
    5.4 We may now have a sufficient number of equations in $S$ to solve for all unknowns in
        $g_f \bmod p$ so attempt this now, calling the result $g_p$.
    5.5 If the system is inconsistent then our original image must be incorrect (missing terms
        or unlucky), so goto 2.
    5.6 If the system is under-determined, then record the degrees of freedom, and if this has
        occurred twice before with the same degrees of freedom then assume that an unlucky
        content problem was introduced by the current prime p so goto 5.1. Otherwise we need
        more images so goto 5.3.1.
    5.7 The resulting solution is consistent and determined, so we have a new image $g_p$
        Set $g_p = \frac{\gamma_p}{\mathrm{lc}_{x_1,...,x_n}(g_p)} \times g_p \bmod p$, $g_m = \mathrm{CRA}([g_p, g_m], [p, m])$, $m = m \times p$.
Until $g_m$ has stopped changing for one iteration.
7 Remove integer content from $g_m$, placing the result in $g_c$, and check that $g_c \mid a$ and $g_c \mid b$.
  If not we need more primes, so goto 5.1.
8 Return $g_c$.
```

Algorithm 2 (LINZIP P)

Input: $a, b \in \mathbb{Z}_p[x_1, ..., x_n]$, *a prime* p, *and degree bounds* $d_{\mathbf{x}}$ *on the* GCD *in* $x_1, ..., x_n$

Output: $g = \mathrm{GCD}(a, b) \in \mathbb{Z}_p[x_1, ..., x_n]$ *or* **Fail**

```
0 Check the GCD of the inputs for content in $x_n$, if present return Fail.
```

```
1 Compute the scaling factor $\gamma = \text{GCD}(\text{lc}_{x_1,\dots,x_{n-1}}(a), \text{lc}_{x_1,\dots,x_{n-1}}(b)) \in \mathbb{Z}_p[x_n]$
2 Choose $v \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\gamma \bmod \langle x_n - v \rangle \neq 0$ and set $a_v = a \bmod \langle x_n - v \rangle$, $b_v = b \bmod \langle x_n - v \rangle$, then compute from these a modular GCD image $g_v \in \mathbb{Z}_p[x_1, \dots, x_{n-1}]$ with a recursive call to LINZIP P $(n > 2)$ or via the Euclidean algorithm $(n = 2)$.
  If for $n > 2$ the algorithm returns **Fail** or for $n = 2$ we have $\deg_{x_1}(g_v) > d_{x_1}$ then return **Fail**, otherwise set $d_{x_1} = \deg_{x_1}(g_v)$ and continue.
3 Assume that $g_v$ has no missing terms, and that the evaluation is not unlucky. We call the assumed form $g_f$. There are two cases here.
  3.1 If there exists a coefficient of $x_1$ in $g_f$ that is a monomial, then we can use single scaling and normalize by setting the integer coefficient of that monomial to 1. Count the largest number of terms in any coefficient of $x_1$ in $g_f$, calling this $n_x$.
  3.2 If there is no such coefficient, then multiple scaling must be used. Compute the minimum number of images needed to determine $g_f$ with multiple scaling, calling this $n_x$.
4 Set $g_{seq} = (\gamma(v)/\text{lc}_{x_1,\dots,x_{n-1}}(g_v)) \times g_v \bmod p$ and $v_{seq} = v$.
5 Repeat
  5.1 Choose a new random $v \in \mathbb{Z}_p \setminus \{0\}$ such that $\gamma \bmod \langle x_n - v \rangle \neq 0$ and set $a_v = a \bmod \langle x_n - v \rangle$, $b_v = b \bmod \langle x_n - v \rangle$.
  5.2 Set $S = \emptyset$, $n_i = 0$.
  5.3 Repeat
    5.3.1 Choose $\alpha_2, \dots, \alpha_{n-1} \in \mathbb{Z}_p \setminus \{0\}$ at random such that for $I = \langle x_2 - \alpha_2, \dots, x_{n-1} - \alpha_{n-1} \rangle$ we have $\deg_{x_1}(a_v \bmod I) = \deg_{x_1}(a)$, $\deg_{x_1}(b_v \bmod I) = \deg_{x_1}(b)$, and set $a_1 = a_v \bmod I$, $b_1 = b_v \bmod I$
    5.3.2 Compute $g_1 = \text{GCD}(a_1, b_1)$
    5.3.3 If $\deg_{x_1}(g_1) < d_{x_1}$ then our original image and form $g_f$ and degree bounds were unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.
    5.3.4 If $\deg_{x_1}(g_1) > d_{x_1}$ our current image $g_1$ is unlucky, so goto 5.3.1, unless the number of failures $> \min(1, n_i)$, in which case assume $x_n = v$ is unlucky and goto 5.1.
    5.3.5 For single scaling, check that the scaling term in the image $g_1$ is present. If not, the assumed form must be wrong, so goto 2.
    5.3.6 Add the equations obtained from equating coefficients of $g_1$ and the evaluation of $g_f \bmod I$ to $S$, and set $n_i = n_i + 1$.
  Until $n_i \geq n_x$
  5.4 We should now have a sufficient number of equations in $S$ to solve for all unknowns in $g_f \bmod p$ so attempt this now, calling the result $g_v$.
  5.5 If the system is inconsistent then our original image must be incorrect (missing terms or unlucky), so goto 2.
  5.6 If the system is under-determined, then record the degrees of freedom, and if this has occurred twice before with the same degrees of freedom then assume the content problem was introduced by the evaluation of $x_n$ so goto 5.1. Otherwise we need more images so goto 5.3.1.
  5.7 The resulting solution is consistent and determined, so we have a new image $g_v$.
      Set $g_{seq} = g_{seq}, \frac{\gamma(v)}{\text{lc}_{x_1,\dots,x_{n-1}}(g_v)} \times g_v$, $v_{seq} = v_{seq}, v$
      Until we have $d_{x_n} + \deg_{x_n}(\gamma) + 1$ images.
  6 Reconstruct our candidate GCD $g_c$ using Newton interpolation (dense) on $g_{seq}$,$v_{seq}$, then removing content in $x_n$.
  7 Probabilistic division test
      Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p$ at random such that for $g_1 = g_c \bmod I$ we have $\deg_{x_1}(g_1) = \deg_{x_1}(g_c)$,
      and compute $a_1 = a \bmod I$, $b_1 = b \bmod I$. Verify that $g_1 \mid a_1$ and $g_1 \mid b_1$, and if not goto 2.
  8 Return $g_c$.
```

We make some remarks before discussing the correctness and termination of the algorithm. For an asymptotic analysis of the *LINZIP* algorithm, the interested reader may consult [12].

1. The degree bound of the GCD in the main variable x_1 is used to detect unlucky primes and evaluations, but only detects those that involve x_1, and we update the degree bound *whenever* we compute a GCD of lower degree in x_1. The degree bounds of the GCD in the non-main variables $x_2, ..., x_n$ are used to compute the number of images needed in the Newton interpolation in step 6 of *LINZIP P*, and are not updated by the algorithm. The degree bound for a variable can be obtained by evaluating the inputs mod a random prime and set of evaluations for all but that variable, then as long as the prime and evaluations are not bad, the univariate GCD provides a bound on the degree of the multivariate GCD for that variable.

2. The number of required images for the multiple scaling case computed in step 3.2 can be the same as the number of required images for the single scaling case computed in step 3.1, and no more than 50% higher. The worst case is quite infrequent, and will only occur when there are only two coefficients with respect to the main variable, each having exactly the same number of terms. The extra expense of this step can usually be reduced by an intelligent choice of the main variable x_1. The exact formula for the number of images needed for a problem with coefficients having term counts of $n_1, ..., n_s$ and a maximum term count of $n_{\max}$ is given by $\max(n_{\max}, \lceil (\sum_{i=1}^{s} n_i - 1)/(s-1) \rceil)$.

3. The check in step 0 of *LINZIP P* is used to detect an unlucky content in the initial GCD introduced higher up in the recursion by either a prime or evaluation. We note that this approach only requires computation of *univariate* contents to detect the problem (if not the source of the problem), as any content in the GCD with respect to x_1 will eventually show up as a univariate content as we evaluate $x_n, x_{n-1}, ...$

4. The check in step 5.6 of either algorithm is intended to check for an unlucky content introduced by the evaluation (*LINZIP P*) or prime (*LINZIP M*) chosen in step 5.1 of both algorithms. Since it is possible that a new random image from step 5.3.1 does not necessarily constrain the form of the GCD (even without the content problem) we check for multiple failures before rejecting the current iteration of loop 5.

5. The *LINZIP P* algorithm performs one probabilistic univariate division test in step 7 instead of testing if $g_c \mid a$ and $g_c \mid b$. This check is substantially less expensive than a multivariate trial division, though there is still a chance that the test fails to detect an incorrect answer, so the termination division test in *LINZIP M* must be retained. Note that to improve the probability that this check detects an incorrect answer, it could be run more than once.

6. Random evaluation points are chosen from $\mathbb{Z}_p \setminus \{0\}$ rather than $\mathbb{Z}_p$ because zero evaluations are likely to cause missing terms in the assumed form, and possibly scaling problems when normalizing images.

To verify the correctness of this algorithm, in addition to the standard issues with modular algorithms we need also verify that the images are scaled consistently to allow the image reconstruction to proceed. We need to consider 4 main problems, namely bad primes or evaluations, unlucky contents, unlucky primes or evaluations, and missing terms in an initial image.

Bad primes and bad evaluations: The treatment of bad primes and bad evaluations is straightforward. It is handled for the first prime or evaluation by the check that γ does not evaluate to 0 in step 2 of the algorithms, handled for subsequent primes or evaluations by the check that γ does not evaluate to 0 in step 5.1 of the algorithms, and handled for the univariate images in step 5.3.1 of the algorithms.

Unlucky content: The unlucky content problem for the first prime or first evaluation is treated in step 0 of *LINZIP P* by the single variable content check. As in point 3 above we emphasize that this check will always detect the problem at some level of the recursion, specifically the level containing the last variable contained in the unlucky content (as all the other variables in the content have been evaluated, so the content becomes univariate). There is no efficient way to detect which prime or evaluation introduced the unlucky content. It may have been introduced by the prime chosen in *LINZIP M* or any evaluation in prior calls (for x_j with $j > n$) to *LINZIP P* in the recursion. Thus we fail all the way back up to the *LINZIP M* algorithm which restarts with a completely new prime and set of evaluations. This strategy is efficient, as only evaluations (modular and variable) and other single variable content checks have been performed before a failure is detected at any level of the recursion.

The introduction of an unlucky content by the prime or evaluation chosen in step 5.1 of either algorithm will be handled in the combination of steps 5.4 and 5.6. The result is a system with additional degrees of freedom, so this *always* results in an under-determined system. The check in step 5.6 handles this, as eventually we will obtain a solution for all variables but the free ones resulting from the unlucky content, so the degrees of freedom will stabilize, and we will go back to step 5.1 choosing a new prime or evaluation.

Unlucky primes and unlucky evaluations: The treatment of unlucky primes and evaluations is less straightforward. First we consider an unlucky evaluation in step 2 of *LINZIP P* for x_n for which the factor added to the GCD depends upon x_1. If the degree bound d_{x_1} is tight, then this will be detected at a lower level of the recursion by step 2 of *LINZIP P* when $n = 2$. If the degree bound d_{x_1} is not tight, then the GCD computed in that step may be unlucky, but we proceed with the computation. Once we reach loop 5, we begin to choose new evaluation points for x_n, and with high probability we will choose a new point that is not unlucky in step 5.1, the problem will be detected in step 5.3.3, and we will go back to step 2, and compute a new image. In the worst case, all evaluations in step 5.1 may also be unlucky, introducing the same factor to the GCD, and we will proceed to step 6, and reconstruct an incorrect result. Note that if the factor is in fact different, then the equations accumulated in step 5.3.5 will most likely be inconsistent, and this problem will most likely be detected in steps 5.4 and 5.5. Step 7 will again perform checks much like those in step 5.3.3, and will detect this problem with high probability, but if it does not, an invalid result may be returned from *LINZIP P*. If we continue to choose unlucky evaluations we will eventually return an incorrect image to *LINZIP M*.

This problem (as well as the unlucky prime case for step 2 of *LINZIP M*) is handled by the structure of the *LINZIP M* algorithm. Since the steps are essentially the same, the same reasoning follows, and we need the computation to be unlucky through all iterations of loop 5. Now in this case, since the form of the GCD is incorrect, it is unlikely that g_m will stabilize, and we will continue to loop. Note that in the event that g_m does stabilize, the invalid image will not divide a and b, so step 7 will put us back into the loop. Now within that loop, which will not terminate until we have found the GCD, step 5.3.4 will eventually detect this problem, as we must eventually find a prime that is not unlucky.

Now consider the case where the unlucky evaluation or prime is chosen in step 2 of either algorithm, and the factor added to the GCD is independent of x_1. In this case, the factor is actually a content with respect to x_1, so this is handled by the same process as the unlucky content problem, specifically it is handled on the way down by step 0 of *LINZIP P*.

Now if an unlucky prime or evaluation occurs in step 5.1 of either algorithm, it will either raise the degree in x_1, in which case it will be detected in step 5.3.4 of either algorithm, or it will be independent of x_1, in which case it is a content. If the content is purely a contribution of the cofactors, then this case will not cause a problem for the algorithm, as it will simply reconstruct the new GCD image without that content present (as a result of the assumed form). The only type of unlucky evaluation that can occur in step 5.3.1 of either algorithm must raise the degree of the GCD in x_1, so is handled by step 5.3.4.

Missing terms: If the initial image (in either algorithm) has missing terms, the resulting system will likely be inconsistent, so will be detected by step 5.5 with high probability, but this may not be the case. If the problem is not detected in any iteration of loop 5, then an incorrect image will be reconstructed in step 6 of *LINZIP P*. The additional check(s) in step 7 of *LINZIP P* will, with high probability, detect this problem with the new images, but if this also fails, then we return an incorrect image from *LINZIP P*. Again assuming a sequence of failures to detect this problem, we arrive at *LINZIP M*. Now we will compute new images in *LINZIP M* until g_c divides both a and b, so the problem must eventually be detected.

Note that the missing term case is the most likely failure case, as unlucky primes, unlucky evaluations, and unlucky contents are in general much less likely. The probability of choosing a prime or evaluation that causes a term to vanish is $\mathcal{O}(\frac{t}{p})$, where t is the number of terms in the polynomial, and p is the prime. Thus the primes used by the algorithm need to be much larger than the number of terms.

4 Algorithm RATZIP

An alternative way of handling the non-monic case is to use sparse rational function interpolation. The idea is as follows. Suppose we are are computing the GCD of two polynomials in $\mathbb{Z}[x, w, y, z]$ with x as the main variable. We will compute the monic GCD in $\mathbb{Z}(w, y, z)[x]$ in the form:

$$x^n + \sum_{i=0}^{n-1} \frac{a_i(w, y, z)}{b_i(w, y, z)} x^i,$$

where $a_i, b_i \in \mathbb{Z}[w, y, z]$ by interpolating the rational function coefficients using a sparse interpolation. For example, if our GCD is $(y + 14)yx^3 + 12y^2x + y + 14$, we compute the monic GCD

$$x^3 + \frac{12y}{y + 14} x + \frac{1}{y}.$$

We then recover the non-monic GCD by multiplying through by the least common multiple of the denominators. In our example, we multiply through by $\text{LCM}(y + 14, y) = (y + 14)y$ to get our non-monic GCD $(y + 14)yx^3 + 12y^2x + y + 14$.

To illustrate how sparse rational function reconstruction works in general, suppose one of the rational function coefficients is $C = \frac{*w^3+*zy^2}{*z^2+*y^2+wy^3}$, here $*$ indicates an integer. Suppose we have reconstructed C at $w = 5$ to get $C_1 = \frac{*+*zy^2}{*z^2+*y^2+y^3}$. Notice we have normalized the leading coefficient of the denominator to be 1, essentially dividing through by w. We then assume the form to be $C_f = \frac{\alpha(w)+\beta(w)zy^2}{\delta(w)z^2+\gamma(w)y^2+y^3}$, where $\alpha(w), \beta(w), \delta(w), \gamma(w)$ are rational functions in w. We have 4 unknowns so we need 4 equations to solve for the next image, C_2. We do this for as many w values as we need, then perform rational function interpolation in w to obtain $\frac{*w^2+\frac{*}{w}zy^2}{\frac{*}{w}z^2+\frac{*}{w}y^2+y^3}$. Clearing the fractions in w gets us what we want, namely $\frac{*w^3+*zy^2}{*z^2+*y^2+wy^3}$.

Example 6 *Let $G, A, B \in \mathbb{Z}[x, y]$ be defined as follows*

$$G = (y + 14)yx^3 + 12y^2x + y + 14, \quad A = (yx + 1)\,G, \quad \textit{and} \quad B = (yx + 2)\,G.$$

Using $p_1 = 11$ we compute our first monic GCD *image in $\mathbb{Z}_{11}(y)[x]$ using dense rational function interpolation. Given a degree bound in y, $d_y = 2$, we need $N = 2d_y + 1 = 5$ evaluation points to interpolate a rational function of the form $\frac{ay^2+by+c}{dy^2+ey+f}$ in y. If we do this by constructing a linear system, the rational function interpolation will cost $\mathcal{O}(N^3)$. Instead we use the Euclidean Algorithm. We first apply the Chinese Remainder Theorem to reconstruct polynomial coefficients in y followed by rational function reconstruction (see [2]). This reduces the cost to $\mathcal{O}(N^2)$. We choose $y = 1, 4, 9, 3, 6$, to get the* GCD *images in $\mathbb{Z}_{11}[x]$, $x^3 + 3x + 1$, $x^3 + 10x + 3$, $x^3 + 9x + 5$, $x^3 + 6x + 4$ and $x^3 + 8x + 2$, respectively. We interpolate in y to get $x^3 + (6y^4 + 9y^3 + 9y^2 + 10y + 2)x + 10y^4 + y^3 + 5y^2 + 3y + 4$ and then apply rational function reconstruction to the coefficients of x to get our first monic* GCD *image $G_1 = x^3 + \frac{y}{y+3}x + \frac{1}{y} \in \mathbb{Z}_{11}(y)[x]$, and our assumed form $G_f = x^3 + \frac{\alpha y}{y+\beta}x + \frac{\delta}{y}$.*

Working modulo $p_2 = 13$ we compute a second monic GCD *image in $\mathbb{Z}_{13}(y)[x]$ using sparse rational function interpolation. We have at most two unknowns per coefficient in our main variable x so we need two evaluation points. We evaluate at $y = 1, 6$, and compute the univariate* GCD *images in $\mathbb{Z}_{13}[x]$, $x^3 + 6x + 1$ and $x^3 + x + 11$, respectively. We evaluate G_f at our chosen y values and equate by coefficient to get the following system.*

$$\left.\begin{array}{rclrcl} 6 &=& \frac{\alpha}{1+\beta}, & 1 &=& \frac{\delta}{1} \\ 1 &=& \frac{6\,\alpha}{6+\beta}, & 11 &=& \frac{\delta}{6} \end{array}\right\} \Rightarrow \alpha = 12, \beta = 1, \delta = 1$$

Substituting back into G_f we get our second monic image in $\mathbb{Z}_{13}(y)[x]$, $G_2 = x^3 + \frac{12\,y}{y+1}x + \frac{1}{y}$.

We then apply the Chinese Remainder Theorem to the integer coefficients of the rational functions of G_1 and G_2 to reconstruct our monic GCD *in $\mathbb{Z}(y)[x]$, $x^3 + \frac{12\,y}{y+14}x + \frac{1}{y}$. Clearing fractions gives us our non-monic* GCD *in $\mathbb{Z}[x, y]$, $(y + 14)yx^3 + 12y^2x + y + 14$.*

We now present the *RATZIP M* algorithm, which computes the GCD in $\mathbb{Z}[x_1, ..., x_n]$ from a number of images in $\mathbb{Z}_p(x_2, ..., x_n)[x_1]$, and the *RATZIP P* algorithm, which computes the GCD in $\mathbb{Z}_p(x_2, ..., x_n)[x_1]$ from a number of images in $\mathbb{Z}_p(x_2, ..., x_{n-1})[x_1]$. As for the *LINZIP M* algorithm any content of the GCD with respect to x_1 must be removed before the initial call to the *RATZIP M* algorithm, and content belonging only to the cofactors can be safely ignored. Unlike in the *LINZIP* algorithms, we do not use single scaling. It is plausible that it may be applied here but it is not straightforward and we have yet to work out the details. The *RATZIP* algorithms are very similar to the *LINZIP* algorithms. The differences are highlighted in shaded boxes. The algorithm is sufficiently similar to the *LINZIP* algorithm so that the treatment of the bad/unlucky primes/evaluations; unlucky contents; and missing terms, applies here without modification.

Algorithm 3 (RATZIP M)

```
Input: $a, b \in \mathbb{Z}[x_1, ..., x_n]$ such that $\text{GCD}(\text{cont}_{x_1}(a), \text{cont}_{x_1}(b)) = 1$ and degree bounds $d_{\mathbf{x}}$ on the
    GCD in $x_1, ..., x_n$
Output: $g = \text{GCD}(a, b) \in \mathbb{Z}[x_1, ..., x_n]$

  1 Compute the scaling factor $\gamma = \text{GCD}(\text{lc}_{x_1,...,x_n}(a), \text{lc}_{x_1,...,x_n}(b)) \in \mathbb{Z}$  (for detection of bad primes)
  2 Choose a random prime $p$ such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$, $b_p =$
    $b \bmod p$, then compute from these a modular GCD image $g_p \in \mathbb{Z}_p(x_2, ..., x_n)[x_1]$ with a call
    to RATZIP P. If the algorithm returns Fail, repeat, otherwise set $d_{x_1} = \deg_{x_1}(g_p)$ and
    continue.
  3 Assume that $g_p$ has no missing terms, and that the prime is not unlucky. We call the assumed
    form $g_f$.

      For each coefficient of $x_1$ in $g_f$, count the number of terms in the numerator nt and
      the number of terms in the denominator dt. Take the maximum sum $nt + dt$ over all
      coefficients and set $n_x = nt + dt - 1$. The $-1$ here is because we normalize the leading
      coefficients of the denominators to be 1.

  4 Set $g_m = g_p$ and $m = p$.
  5 Repeat
    5.1 Choose a new random prime $p$ such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$,
        $b_p = b \bmod p$.
    5.2 Set $S = \emptyset$, $n_i = 0$.
    5.3 Repeat
      5.3.1 Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p \backslash \{0\}$ at random such that for $I = \langle x_2 - \alpha_2, ..., x_n - \alpha_n \rangle$ we have
            $\deg_{x_1}(a_p \bmod I) = \deg_{x_1}(a)$, $\deg_{x_1}(b_p \bmod I) = \deg_{x_1}(b)$, and set $a_1 = a_p \bmod I$,
            $b_1 = b_p \bmod I$
      5.3.2 Compute $g_1 = \text{GCD}(a_1, b_1)$
      5.3.3 If $\deg_{x_1}(g_1) < d_{x_1}$ then our original image and form $g_f$ and degree bounds were
            unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.
      5.3.4 If $\deg_{x_1}(g_1) > d_{x_1}$ our current image $g_1$ is unlucky, so goto 5.3.1, unless the number
            of failures $> \min(2, n_i)$, in which case assume $p$ is unlucky and goto 5.1.
      5.3.5 Add the equations obtained from equating coefficients of $g_1$ and the evaluation of
            $g_f \bmod I$ to $S$, and set $n_i = n_i + 1$.
        Until $n_i \geq n_x$
    5.4 We may now have a sufficient number of equations in $S$ to solve for all unknowns in
        $g_f \bmod p$ so attempt this now, calling the result $g_p$.
    5.5 If the system is inconsistent then our original image must be incorrect (missing terms
        or unlucky), so goto 2.
    5.6 If the system is under-determined, then record the degrees of freedom, and if this has
        occurred twice before with the same degrees of freedom then assume that an unlucky
```

content problem was introduced by the current prime p so goto 5.1. Otherwise we need more images so goto 5.3.1.

5.7 *The resulting solution is consistent and determined, so we have a new image g_p.*

Apply the Chinese remainder theorem to update g_m by combining the coefficients of $g_p \in \mathbb{Z}_p(x_2, ..., x_n)[x_1]$ with $g_m \in \mathbb{Z}_m(x_2, ..., x_n)[x_1]$,

updating $m = m \times p$.

5.8 *Apply rational reconstruction to the integer coefficients of g_m, then clear the fractions to get $g \in \mathbb{Z}(x_2, ..., x_n)[x_1]$.*

Until g has stopped changing for one iteration.

7 *Clear the rational function denominators of g to get $g \in \mathbb{Z}[x_1, ..., x_n]$. Compute $g_c = \mathrm{pp}_{x_1,...,x_n}(g)$ and check that $g_c \mid a$ and $g_c \mid b$. If not we need more primes, so goto 5.1.*

8 *Return g_c.*

Note that in step 7 of *RATZIP M*, clearing the denominators involves a series of multivariate lowest common multiple computations, which could potentially be expensive.

Algorithm 4 (RATZIP P)

Input: *$a, b \in \mathbb{Z}_p[x_1, ..., x_n]$, a prime p, and degree bounds $d_{\mathbf{x}}$ on the* GCD *in $x_1, ..., x_n$*

Output: $g = \mathrm{GCD}(a, b) \in \mathbb{Z}_p(x_2, ..., x_n)[x_1]$ *or* **Fail**

0 *Check the* GCD *of the inputs for content in x_n, if present return* **Fail**.

1 *Compute the scaling factor $\gamma = \mathrm{GCD}(\mathrm{lc}_{x_1,...,x_{n-1}}(a), \mathrm{lc}_{x_1,...,x_{n-1}}(b)) \in \mathbb{Z}_p[x_n]$.*
If $\gamma = 1$ then set $RR = False$ else set $RR = True$.

2 *Choose $v \in \mathbb{Z}_p$ at random such that $\deg_{x_1,...,x_{n-1}}(a \bmod \langle x_n - v\rangle) = \deg_{x_1,...,x_{n-1}}(a)$, $\deg_{x_1,...,x_{n-1}}(b \bmod \langle x_n - v\rangle) = \deg_{x_1,...,x_{n-1}}(b)$, and set $a_v = a \bmod \langle x_n - v\rangle$, $b_v = b \bmod \langle x_n - v\rangle$, then compute from these a modular* GCD *image $g_v \in \mathbb{Z}_p(x_2, ..., x_{n-1})[x_1]$ with a recursive call to RATZIP P $(n > 2)$ or via the Euclidean algorithm $(n = 2)$.*
If for $n > 2$ the algorithm returns **Fail** *or for $n = 2$ we have $\deg_{x_1}(g_v) > d_{x_1}$ then return* **Fail**, *otherwise set $d_{x_1} = \deg_{x_1}(g_v)$ and continue.*

3 *Assume that g_v has no missing terms, and that the evaluation is not unlucky. We call the assumed form g_f.*

For each coefficient of x_1 in g_f, count the number of terms in the numerator nt and the number of terms in the denominator dt. Take the maximum sum $nt + dt$ over all coefficients and set $n_x = nt + dt - 1$. The -1 here is because we normalize the leading coefficients of the denominators to be 1.

4 *Set $g_m = g_v$, $m = x_n - v$, and $N_i = 1$.*

5 *Repeat*

```
    5.1 Choose a new random $v \in \mathbb{Z}_p$ such that $\deg_{x_1,..,x_{n-1}}(a \bmod \langle x_n - v\rangle)$
        $= \deg_{x_1,..,x_{n-1}}(a)$, $\deg_{x_1,..,x_{n-1}}(b \bmod \langle x_n - v\rangle) = \deg_{x_1,..,x_{n-1}}(b)$, and
        set $a_v = a \bmod \langle x_n - v\rangle$, $b_v = b \bmod \langle x_n - v\rangle$.
    5.2 Set $S = \emptyset$, $n_i = 0$.
    5.3 Repeat
      5.3.1 Choose $\alpha_2, ..., \alpha_{n-1} \in \mathbb{Z}_p$ at random such that for $I = \langle x_2 - \alpha_2, ..., x_{n-1} - \alpha_{n-1}\rangle$
            we have $\deg_{x_1}(a_v \bmod I) = \deg_{x_1}(a)$, $\deg_{x_1}(b_v \bmod I) = \deg_{x_1}(b)$, and set $a_1 =$
            $a_v \bmod I$, $b_1 = b_v \bmod I$
      5.3.2 Compute $g_1 = \mathrm{GCD}(a_1, b_1)$
      5.3.3 If $\deg_{x_1}(g_1) < d_{x_1}$ then our original image and form $g_f$ and degree bounds were
            unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.
      5.3.4 If $\deg_{x_1}(g_1) > d_{x_1}$ our current image $g_1$ is unlucky, so goto 5.3.1, unless the number
            of failures $> \min(1, n_i)$, in which case assume $x_n = v$ is unlucky and goto 5.1.
      5.3.5 Add the equations obtained from equating coefficients of $g_1$ and the evaluation of
            $g_f \bmod I$ to $S$, and set $n_i = n_i + 1$.
      Until $n_i \geq n_x$
    5.4 We should now have a sufficient number of equations in $S$ to solve for all unknowns in
        $g_f \bmod p$ so attempt this now, calling the result $g_v$.
    5.5 If the system is inconsistent then our original image must be incorrect (missing terms
        or unlucky), so goto 2.
    5.6 If the system is under-determined, then record the degrees of freedom, and if this has
        occurred twice before with the same degrees of freedom then assume the content problem
        was introduced by the evaluation of $x_n$ so goto 5.1. Otherwise we need more images so
        goto 5.3.1.
    5.7 The resulting solution is consistent and determined, so we have a new image $g_v$.

        Solve $f \equiv g_m \pmod{m(x_n)}$ and $f \equiv g_v \pmod{x_n - v}$ using the Chinese remainder
        algorithm for $f \in \mathbb{Z}_p[x_n](x_2, ..., x_{n-1})[x_1] \bmod m(x_n) \times (x_n - v)$.
        Set $g_m = f, m = m(x_n) \times (x_n - v)$, and $N_i = N_i + 1$.

  Until $N_i \geq d_{x_n} + 1$ and ($RR = False$ or $N_i \geq 3$)
6 Reconstruct
      6 If $RR = True$ then apply rational function reconstruction in $x_n$ and assign the
        result to $g_c$. For $n > 2$, clear the rational function denominators of $g_c \in$
        $\mathbb{Z}_p(x_n)(x_2, ..., x_{n-1})[x_1]$ to obtain $g_c \in \mathbb{Z}_p(x_2, ..., x_n)[x_1]$. If rational function recon-
        struction fails then we need more points, goto 5.1.
    6.2 If $RR = False$ then set $g_c = g_m$.
7 Probabilistic division test
  Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p$ at random such that for $g_1 = g_c \bmod I$ we have $\deg_{x_1}(g_1) = \deg_{x_1}(g_c)$,
  and compute $a_1 = a \bmod I$, $b_1 = b \bmod I$. Verify that $g_1 \mid a_1$ and $g_1 \mid b_1$, and if not goto 2.
8 Return $g_c$.
```

5 Implementation

We have implemented both algorithms in Maple using the "recden" [7] data structure. Because it supports multiple field extensions over $\mathbb{Q}$ and $\mathbb{Z}_p$, it will allow us to extend our implementations to work over finite fields and algebraic number fields. The linear algebra over $\mathbb{Z}_p$ and univariate polynomial computations over $\mathbb{Z}_p$ and the integer arithmetic are all coded in C. The rest is coded in Maple. The data structure is currently being implemented in the kernel of Maple for improved efficiency.

We present timings on three classes of test problems. All problems were run on a Pentium 4 3.0 GHz processor with 1 GB of RAM and all times are given in CPU seconds.

Balanced Sparse Problems

These problems are in $\mathbb{Z}[x, y, z, w, t]$, with x chosen to be the main variable. The GCD and cofactors are random polynomials of maximum degree d in each variable, with $2d$ terms, and integer coefficients taken from the range 100 to 1000. We vary d from 3 to 10.

d	Total Number of Univariate GCDs		Total Time		Time Ratio
	LINZIP	*RATZIP*	*LINZIP* (t_1)	*RATZIP* (t_2)	(t_1 / t_2)
3	62	73	0.55	0.73	2.75
4	85	94	1.09	1.49	0.73
5	147	112	2.42	2.15	1.13
6	178	148	4.04	4.33	0.93
7	219	159	6.97	5.52	1.26
8	187	127	8.46	7.48	1.13
9	340	280	17.20	14.55	1.18
10	325	271	24.62	21.89	1.12

Table 1: Run time results for balanced sparse GCDs

Balanced Semi-Sparse Problems

These problems are intended to reflect a typical class of problems frequently encountered in realistic applications. The polynomials are in $\mathbb{Z}[x, y, z, w, t]$, with x chosen to be the main variable. The GCD and cofactors are random polynomials of maximum degree d in each variable, with $\sqrt{(d+1)^5}$ terms, and integer coefficients taken from the range 100 to 1000. We vary d from 3 to 8.

d	Terms	Total Number of Univariate GCDs		Total Time		Time Ratio
		LINZIP	*RATZIP*	*LINZIP* (t_1)	*RATZIP* (t_2)	(t_1 / t_2)
3	32	214	303	5.59	8.34	0.67
4	56	396	541	21.00	33.66	0.62
5	89	466	754	56.92	98.11	0.58
6	130	583	1275	143.57	245.92	0.58
7	182	977	1490	478.84	834.84	0.57
8	243	1237	1783	1039.92	1834.02	0.57

Table 2: Run time results for balanced semi-sparse GCDs

Sparse Problems with Extraneous Leading Coefficient GCD

These test problems are constructed such that the GCD of leading coefficients of the input polynomials is larger than the leading coefficient of the actual GCD . These problems are in $\mathbb{Z}[x, y, z, w, t]$, with x chosen to be the main variable. We take $f = (x^d + a)^2((y^d z^d w^d t^d + b)x^d + c)$, where a, b and c are random polynomials in $\mathbb{Z}[y, z, w, t]$ of total degree $d - 1$ with d terms and integer coefficients ranging from 100 to 1000. We impose the condition that the $\text{GCD}(y^d z^d w^d t^d + b, c) = 1$, and we vary d from 3 to 10. We compute the GCD of f and $\frac{\partial f}{\partial x}$ which is $x^d + a$. Note that the GCD of the leading coefficients of the input polynomials is $y^d z^d w^d t^d + b$. We note that this problem set is a highly specialized case designed to show the behavior of the algorithms when the rational function approach provides performance superior to the modified normalization approach.

d	Total Number of Univariate GCDs		Total Time		Time Ratio
	LINZIP	*RATZIP*	*LINZIP* (t_1)	*RATZIP* (t_2)	(t_1 / t_2)
3	37	20	0.20	0.17	1.18
4	65	37	0.40	0.34	1.17
5	118	67	1.16	0.93	1.25
6	148	74	1.70	1.29	1.32
7	209	108	2.87	2.12	1.35
8	294	141	5.16	3.45	1.50
9	312	137	5.67	3.79	1.50
10	381	175	9.27	6.57	1.41

Table 3: Run time results for sparse GCDs with extraneous leading coefficient GCD

Final Remarks

Our implementation of the *RATZIP* algorithm includes the following enhancement. To reconstruct the rational functions in some variable y with degree bound d_y, we need $2d_y + 1$ evaluation points. In fact, we may need fewer points than this, depending on the form of the rational functions being reconstructed. In our implementation we are using the *Maximal Quotient Rational Reconstruction* algorithm [8], which uses at most one more evaluation point than the minimum number of points required for the reconstruction to succeed. For example, to reconstruct the rational functions in y of $G = x^3 + \frac{y}{y+3}x + \frac{1}{y}$ from Example 6, we would need 4 points, not 5.

A disadvantage of Zippel's algorithm is the large number of univariate images that must be computed for the sparse interpolations. Most of the time is usually in the evaluations in step 5.3.1 and not the univariate GCD computations in step 5.3.2. For the test problems presented in section 5, the percentage of time spent on evaluations was on average 68% and 75% for *LINZIP* and *RATZIP*, respectively. The multivariate trial division in step 7 of *LINZIP M* and *RATZIP M* took 19% and 11% of the time, respectively. Together the evaluations and the division cost comprise over 86% of the total time.

To improve the efficiency, instead of evaluating out all but one variable x_1, consider evaluating out all but 2 variables x_1, x_2 and computing the bivariate images using a dense GCD algorithm. Thus think of G as a polynomial in x_1 and x_2 (main variables) with coefficients in $\mathbb{Z}[x_3, ..., x_n]$. If G is dense in x_1 and x_2 then little efficiency is lost. We gain a likely significant reduction in t the maximum number of terms of the coefficients in x_1 and x_2, hence a reduction in the maximum size of the linear systems and a reduction in the number of images needed for the sparse interpolations. We also increase the likelihood of not needing to apply the multiple scaling or rational reconstruction methods. Furthermore, we simplify the multivariate GCD computation for the content of G and, in RATZIP, the final LCM computation.

References

[1] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* **18** (1971), 478-504.

[2] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, UK, 1999.

[3] E. Kaltofen. Sparse Hensel lifting. *Proceedings of EUROCAL 85*, Springer-Verlag LNCS **2**, pages 4-17, 1985.

[4] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comp.* **9** (1990), 301-320.

[5] E. Kaltofen, W. Lee, A. Lobo. Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's Algorithm. *Proceedings of ISSAC 2000*, ACM Press, (2000), 192–201

[6] E. Kaltofen, W. Lee. Early Termination in Sparse Interpolation Algorithms. *J. Symbolic Comp.* **36** (3-4) (2003), 365–400.

[7] M. van Hoeij, M. B. Monagan. A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *Proceedings of ISSAC '2002*, ACM Press, pp. 109–116, 2002.

[8] M. B. Monagan. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '2004*, ACM Press, 243–249, 2004.

[9] P. S. Wang. An Improved Multivariate Polynomial Factorization Algorithm. *Mathematics of Computation* **32**, pp. 1215–1231, 1978.

[10] P. S. Wang. The EEZ-GCD algorithm. *ACM SIGSAM Bull.* **14**, (1980), 50-60.

[11] M. O. Rayes, P. S. Wang, K. Weber. Parallelization of The Sparse Modular GCD Algorithm for Multivariate Polynomials on Shared Memory Multiprocessors, *Proceedings of ISSAC '94*, ACM Press, pp. 66-73, 1994.

[12] A. D. Wittkopf, Algorithms and Implementations for Differential Elimination, Ph.D. Thesis, Simon Fraser Univ. (2004) (`http://www.cecm.sfu.ca/~wittkopf/WittThesis.pdf`).

[13] R. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proceedings of EUROSAM '79*, Springer-Verlag LNCS, **2** pp. 216–226, 1979.

[14] R. Zippel. Interpolating Polynomials from their Values. *J. Symbolic Comput.* **9**, 3 (1990), 375-403.

[15] R. Zippel, *Effective Polynomial Computation*, Kluwer Academic, 1993.

Appendix

We describe how to remove the content of G from the inputs A, B. Let $G = \sum_{i=0}^{d} g_i x_1^i$, $A = \sum_{i=0}^{m} a_i x_1^i$, and $B = \sum_{i=0}^{n} b_i x_1^i$. Recall that the content C of G in x_1 is $\gcd(g_0, g_1, ..., g_d)$. The obvious way to compute C is to compute the GCD of the contents of A and B, that is, to compute $C = \gcd(\gcd(\{a_i\}), \gcd(\{b_i\}))$ and set $A = A/C$ and $B = B/C$. This should not be done in this manner, as it requires at least three (probably multivariate) GCD computations, and is expensive when the contents of the inputs A and B are larger than the content of G. Indeed, if $G = 1$ computing the contents of A and B would make the overall algorithm no longer output sensitive to the size of G. Instead, select the coefficient of A and B of smallest size and compute H the GCD of that coefficient and a *random linear combination* of all other coefficients. Then verify that H divides the other coefficients.

That our algorithms must compute and divide out by the content of G may introduce an inefficiency. For example, take $G = (y^p - 1)x^5 + (y^q - 1)$ for primes $p \neq q$. Then $C = y - 1$ and $G/C = (y^{p-1} + ... + y + 1)x^5 + (y^{q-1} + ... + y + 1)$ is dense in y. In practice this is not a serious problem. If the content of G is 1, the recursive GCD will determine this quickly because it is using a modular GCD algorithm, and hence, little time is wasted. If the content C of G is non-trivial, then most likely G/C will have fewer terms than G, not more, and as a result the main GCD computation will complete more rapidly.

We mention that Kaltofen and Trager's black box approach in [4] solves both the normalization problem and the content problem by a clever change of variables. Replace each non-main variable x_i in the input by $x_1 + \alpha x_i$ for some random integer α. Notice that this makes the inputs monic in the main variable x_1. Thus the modified input has no content (it has become part of the main GCD) and there is no normalization problem. In the black-box model these substitutions cost essentially nothing but in the standard expanded representations for polynomials they make the inputs dense, hence we do not do this.